Docket No. AUS920000752US1

CACHE THRESHOLDING METHOD, APPARATUS, AND PROGRAM FOR PREDICTIVE REPORTING OF ARRAY BIT LINE OR DRIVER FAILURES

BACKGROUND OF THE INVENTION

1. Technical Field:

The present invention relates to data processing and, in particular, to error detection and correction. Still more particularly, the present invention provides a method, apparatus, and program for predicting array bit line or driver failures.

2. Description of Related Art:

A system may include an event scan that is invoked periodically for each processor in the system. The system may also include error correction circuitry (ECC) to resolve correctable single-bit errors (CE). Some of these correctable errors may be detected in processor caches. A CPU Guard function can be used to dynamically de-allocate a processor and cache that has an error. A Repeat Guard function can be used to de-allocate the resource during boot process to ensure that the cache with the fault does not cause further errors until a customer engineer is able to fix the error. The system may include field replaceable units (FRU), each of which includes a processor and cache. The customer engineer may fix a fault by replacing the FRU.

A cache may have array bit line or driver failures that may cause correctable errors to be detected. Even though these errors are corrected and do not impact continued operation, it is desirable to detect when CEs

are repeatedly caused by these types of cache faults. Prior art algorithms attempt to detect array bit line or driver failures by simply counting to some number of faults within a specified time period. Within those algorithms, however, intermittent single-bit errors caused by random noise or other cosmic conditions may result in many false reports of bit line or driver failures.

In addition, some prior algorithms rely on the system rebooting periodically to reset error threshold counters. However, in today's business-critical computing environments, computer systems are not rebooted very often. This may cause random errors to accumulate and trigger false reports.

Thus, it would be advantageous to provide a cache thresholding method and apparatus for predictive reporting of array bit line or driver failures that does not generate false error reports because of random errors.

SUMMARY OF THE INVENTION

The present invention provides a mechanism for predicting cache array bit line or driver failures, which is faster and more efficient than counting all of the errors associated with a failure. This mechanism checks for five consecutive errors at different addresses within the same syndrome on invocation of periodic polling to characterize the failure. Once the failure is characterized, it is reported to the system for corrective maintenance including dynamic processor deconfiguration or preventive processor replacement.

BRIEF DESCRIPTION OF THE DRAWINGS

The novel features believed characteristic of the invention are set forth in the appended claims. The invention itself, however, as well as a preferred mode of use, further objectives and advantages thereof, will best be understood by reference to the following detailed description of an illustrative embodiment when read in conjunction with the accompanying drawings, wherein:

Figure 1 is a block diagram of an illustrative embodiment of a data processing system with which the present invention may advantageously be utilized;

Figure 2 is a diagram of an error table in accordance with a preferred embodiment of the present invention;

Figure 3 is a flowchart illustrating a system initialization process in accordance with a preferred embodiment of the present invention; and

Figure 4 is a flowchart of the operation of the process for predicting bit line or driver failures in accordance with a preferred embodiment of the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

Referring now to the drawings and in particular to **Figure 1**, there is depicted a block diagram of an illustrative embodiment of a data processing system with which the present invention may advantageously be utilized. As shown, data processing system **100** includes processor cards **111a-111n**. Each of processor cards **111a-111n** includes a processor and a cache memory. For example, processor card **111a** contains processor **112a** and cache memory **113a**, and processor card **111n** contains processor **112n** and cache memory **113n**.

Processor cards **111a-111n** are connected to main bus **115**. Main bus **115** supports a system planar **120** that contains processor cards **111a-111n** and memory cards **123**. The system planar also contains data switch **121** and memory controller/cache **122**. Memory controller/cache **122** supports memory cards **123** that includes local memory **116** having multiple dual in-line memory modules (DIMMs).

Data switch **121** connects to bus bridge **117** and bus bridge **118** located within a native I/O (NIO) planar **124**. As shown, bus bridge **118** connects to peripheral components interconnect (PCI) bridges **125** and **126** via system bus **119**. PCI bridge **125** connects to a variety of I/O devices via PCI bus **128**. As shown, hard disk **136** may be connected to PCI bus **128** via small computer system interface (SCSI) host adapter **130**. A graphics adapter **131** may be directly or indirectly connected to PCI bus **128**. PCI bridge **126** provides connections for external

data streams through network adapter **134** and adapter card slots **135a-135n** via PCI bus **127**.

An industry standard architecture (ISA) bus **129** connects to PCI bus **128** via ISA bridge **132**. ISA bridge **132** provides interconnection capabilities through NIO controller **133** having serial connections Serial 1 and Serial 2. A floppy drive connection **137**, keyboard connection **138**, and mouse connection **139** are provided by NIO controller **133** to allow data processing system **100** to accept data input from a user via a corresponding input device. In addition, non-volatile RAM (NVRAM) **140** provides a non-volatile memory for preserving certain types of data from system disruptions or system failures, such as power supply problems. A system firmware **141** is also connected to ISA bus **129** for implementing the initial Basic Input/Output System (BIOS) functions. A service processor **144** connects to ISA bus **129** to provide functionality for system diagnostics or system servicing.

The operating system (OS) is stored on hard disk **136,** which may also provide storage for additional application software for execution by data processing system. NVRAM **140** is used to store system variables and error information for field replaceable unit (FRU) isolation. During system startup, the bootstrap program loads the operating system and initiates execution of the operating system. To load the operating system, the bootstrap program first locates an operating system kernel type from hard disk **136**, loads the OS into memory, and jumps to an initial address provided by the operating

system kernel. Typically, the operating system is loaded into random-access memory (RAM) within the data processing system. Once loaded and initialized, the operating system controls the execution of programs and may provide services such as resource allocation, scheduling, input/output control, and data management.

The present invention may be executed in a variety of data processing systems utilizing a number of different hardware configurations and software such as bootstrap programs and operating systems. The data processing system **100** may be, for example, a stand-alone system or part of a network such as a local-area network (LAN) or a wide-area network (WAN).

The preferred embodiment of the present invention, as described below, is implemented within a data processing system **100** in which an event scan is invoked periodically for each processor. The event scan stores an error table, an error position pointer, and an error counter for each processor.

With reference now to **Figure 2**, a diagram of an error table is shown in accordance with a preferred embodiment of the present invention. The error table **200** for processor n level two (L2) cache includes a column for the address of the error **202** and a column for the correctable error (CE) flag **204**. The CE error flag is set if an error is repeating during error polling period. This is an indication of a solid error. If the error address is unique during each polling period, the error address/ syndrome entry is saved in the next error entry. A syndrome is an error signature that describes the

error. For example, for a block of sixteen bytes read, the syndrome may identify which bit is bad.

If there are five error entries, all syndromes are checked for error bit alignment. If all five error syndromes are equal, the event scan reports a CE threshold condition. After the fifth error entry is saved and all syndromes are not equal, the next entry restarts at the first error position.

With reference now to **Figure 3**, a flowchart is shown illustrating a system initialization process in accordance with a preferred embodiment of the present invention. The process begins and sets standby power on (step **302**). The process then performs service processor initialization (step **304**) and sets system power on (step **306**). Thereafter, the process performs service processor hardware verification and initialization (step **308**). System initialization is performed by system firmware (step **310**) and the process boots AIX (step **312**). During system initialization, tables are cleared and counters are set to zero. Finally, the process begins periodic event scan calls (step **314**) and ends.

Turning now to **Figure 4**, a flowchart of the operation of the process for predicting bit line or driver failures is shown in accordance with a preferred embodiment of the present invention. The process begins with an event scan call for a processor. The process then reads the L2 cache status register (L2SR) (step **402**) and a determination is made as to whether a cache CE is detected (step **404**). If a cache CE is not detected, the process clears the error flag and all addresses in the

error table and sets the error counter to (step **406**). Then, the process proceeds to step **418** to end event scan processing.

If a cache CE is detected in step **404**, a determination is made as to whether the CE flag is set (step **408**). If the CE flag is not set, the process sets the CE flag (step **410**), saves the L2 address and syndrome in the error table (step **412**), and sets the error position pointer to two (step **414**). Then, the process increments the error counter (step **416**) and ends event scan processing (step **418**).

If the CE flag is set in step **408**, a determination is made as to whether the new address is equal to the stored address (step **420**). If the new address does not equal the stored address, the process stores the new address and syndrome in the next error entry (step **422**) and a determination is made as to whether the error position pointer equals five. If the error position pointer equals five, the process sets the error position equal to one (step **426**) and proceeds to step **430** to check if error counter is less than five.

If the error position pointer is not equal to five in step **424**, the process increments the error position pointer (step **428**). Returning to step **420**, if the new address equals the stored address, the process proceeds to step **428** to increment the error position pointer. Thereafter, a determination is made as to whether the error counter is less than five (step **430**). If the error counter is less than five, the process increments the

error counter and ends event scan processing (step **418**).

If the error counter is not less than five in step **430**, then the error counter must be equal to five and the process compares all five syndromes in the error table (step **434**) and a determination is made as to whether all syndromes are the same (step **436**). If all syndromes are not the same, the process ends event scan processing (step **418**). This filters out random single-bit errors.

If all syndromes are the same in step **436**, the process clears the CE error table (step **440**), records the error status and reason code (step **442**), and creates an error log identifying cache predictive failure (step **444**). Next, the process returns the error log to the operating system (step **446**) and ends event scan processing (step **418**).

Thus, the present invention solves the disadvantages of the prior art by filtering out random occurrences of intermittent errors. The present invention reports on bit line or driver faults by insuring that the same syndrome is reported on five different address locations in consecutive checks by the event scan polling code. This reporting eliminates false triggering of predictive maintenance actions which result in either the dynamic de-allocation of the processor module associated with that cache or the unnecessary replacement of the FRU.

The present invention minimizes the risk of uncorrectable errors by invoking dynamic CPU deconfiguration at runtime if enabled, boot-time CPU deconfiguration if enabled for subsequent boots, and preventive FRU replacement. The present invention

provides better system availability and performance by not falsely de-allocating or replacing working cache modules. This, in turn, reduces service and warranty costs and increases customer satisfaction.

It is important to note that while the present invention has been described in the context of a fully functioning data processing system, those of ordinary skill in the art will appreciate that the processes of the present invention are capable of being distributed in the form of a computer readable medium of instructions and a variety of forms and that the present invention applies equally regardless of the particular type of signal bearing media actually used to carry out the distribution. Examples of computer readable media include recordable-type media such a floppy disc, a hard disk drive, a RAM, and CD-ROMs and transmission-type media such as digital and analog communications links.

The description of the present invention has been presented for purposes of illustration and description, but is not intended to be exhaustive or limited to the invention in the form disclosed. Many modifications and variations will be apparent to those of ordinary skill in the art. The embodiment was chosen and described in order to best explain the principles of the invention, the practical application, and to enable others of ordinary skill in the art to understand the invention for various embodiments with various modifications as are suited to the particular use contemplated.